AIR UNIVERSITY

Software Quality Engineering

Lab Task 06

Group No : 2
Section : BSSE – V – A
Date: 25th Oct
Submiited To : Sir Naseer Jan

Contents

1.Group Information

Member Name	*Role*	*Responsibilities*
Muhammad Shamoil	***Moderator***	***Coordinated all members, scheduled/led the inspection meeting, and summarized all findings into the final PDF report.***
Arslan Jaffer	***Reworker***	***Responsible for implementing the "Rework" (fixing all identified defects) based on the Scribe's final defect log***
Ali Raza	***Reader***	***Guided the inspection meeting by reading code sections aloud and created the fixed UML Class Diagram (Section 3.2).***
Muhammad Awais	***Inspector***	***Performed individual preparation. Inspected the code against all provided checklists and SOLID principles to find defects.***
Abdullah Awan	***Analyzer / Scribe***	***Recorded all defects identified during the meeting. Classified each defect's severity and logged it in the official defect sheet (Section 4.1).*** ***Helped Reowrker in Fixing the Defects***

2. Software Inspection Process Overview

2.1. Inspection Process Followed

Our team conducted a formal Software Inspection process following the six standard phases:

Planning: The Moderator (Muhammad Shamoil) assigned roles and scheduled the main inspection meeting via the group chat. All checklists and the defective code were distributed.

Overview: The Moderator gave a brief overview of the project requirements and the expected deliverables.

Preparation: Each team member, especially the Inspector (Abdullah Awan), reviewed the UML diagram and all .java files individually using the checklists before the meeting.

Inspection Meeting: The full team met online. The Reader (Ali Raza) guided the discussion. The Inspector raised issues, which were discussed by the team. The Scribe (Awais) formally logged every confirmed defect.

Rework: After the meeting, the defect log was given to the Reworker (Arslan Jaffer) to fix all identified issues.

Follow-up: The Moderator verified that all logged defects were resolved by the Reworker, confirming the project was in a stable, executable state.

2.2. Meeting and Time Log

Time: 8PM – 9:30PM

Duration: 1.5 hours

Venue/Platform: WhatsApp Group Call

Attendees: Muhammad Shamoil, Arslan Jaffer, Ali Raza, Abdullah Awan, Awais

3.UML Class Diagram Review

3.1. Analysis of Original UML Diagram

3.1.1. Interface Segregation Principle (ISP) Violation

The primary issue is the SpaceCraftOperations interface. This is a "fat interface" because it bundles multiple, unrelated responsibilities into a single contract. As implied by the project description and the various concrete classes, this interface contains methods for:

1. Launching (e.g., launch())
2. Landing (e.g., land())
3. Maneuvering (e.g., executeManeuver())
4. Communicating (e.g., transmitData())

This violation forces any class that implements it to also implement methods it has no use for. **Example:** A ParachuteLanding class only needs to land(). It should not be forced to have methods for launch() or transmitData(). This leads to defective code where these methods are left empty or throw exceptions.

Example: A LaserCommunication class only needs to transmitData(). It has no concept of landing or launching.

3.1.2. Single Responsibility Principle (SRP) Violation

This violation is a direct consequence of the ISP violation.

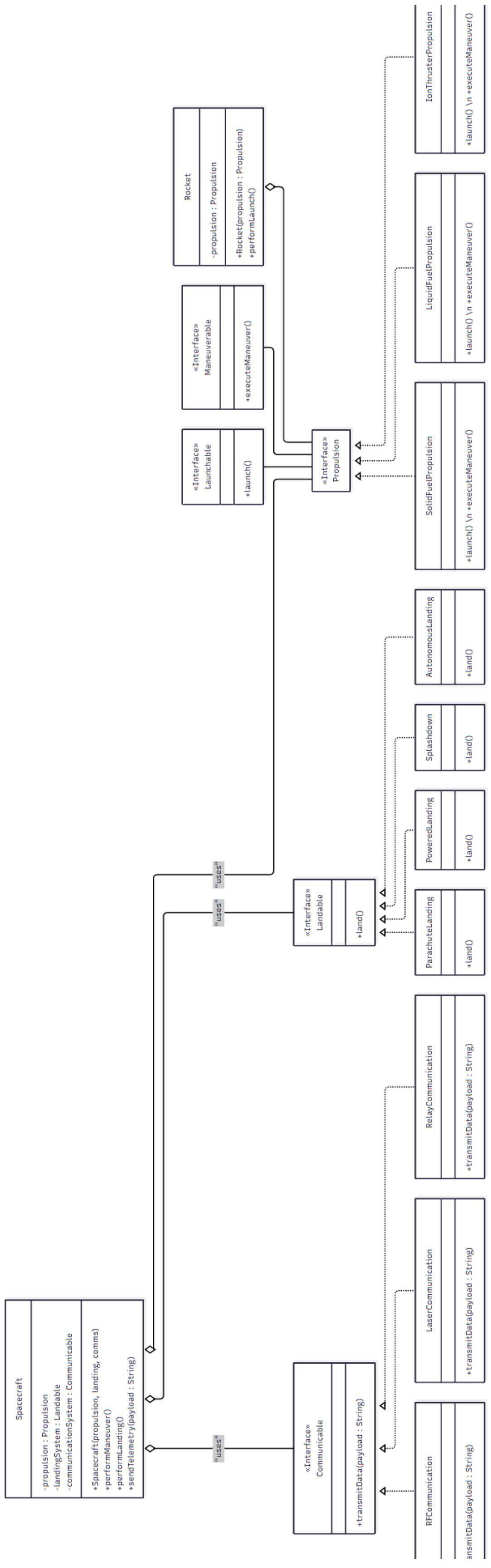
1. **At the Interface Level:** The SpaceCraftOperations interface violates SRP because it is responsible for defining contracts for several different "concerns" (landing, propulsion, communication).
2. **At the Class Level:** Any class implementing this "fat interface" is forced to violate SRP. For instance, a SolidFuelPropulsion class should *only* be responsible for propulsion. The original design, however, forces it to also be responsible for communication and landing, even if those methods are empty.

3.1.3. Dependency Inversion Principle (DIP) & Open/Closed Principle (OCP) Violation: Tight Coupling

The diagram shows high-level classes, like Rocket, having direct associations with low-level, concrete classes like SolidFuelPropulsion or ParachuteLanding.

1. **DIP Violation:** This is a clear violation of the Dependency Inversion Principle, which states that high-level modules should not depend on low-level modules; both should depend on abstractions. The Rocket class should *only* depend on interfaces (abstractions) like Propulsion, Landable, and Communicable, not on the concrete implementations.
2. **OCP Violation:** This tight coupling directly violates the Open/Closed Principle. The Rocket class is not "open for extension" and "closed for modification."
   - **Example:** If we want to add a new LiquidFuelPropulsion type, we would have to modify the Rocket class to accept or instantiate this new concrete class.
   - **The Fix:** By depending on the Propulsion interface (following DIP), we can create a new LiquidFuelPropulsion class and *pass it* to the Rocket without ever changing a single line of code inside Rocket.

3.2. Fixed UML Class Diagram



4. Source Code Inspection

4.1. Defect Log

Defect ID	File Name & Line	Defect Description	Checklist Reference	Severity	SOLID Violation (if any)	Recommended Fix
Critical Defects (Compilation Failures & Crashes)						
D-001	Rocket.java, Line 46	int status = 1 / 0; - Code contains a "Divide by Zero" exception, which will crash the program during performLaunch().	Java Checklist: 4.1 (Bugs)	**Critical**	N/A	Remove the line.
D-002	Rocket.java, Line 60	landingSystem.land(); - This line will not compile. The Landable interface is empty, and concrete classes have incompatible land() signatures.	Custom Checklist: 2.1 (Design)	**Critical**	N/A	1. Add void land(); to Landable. 2. Update all implementing classes to @Override void land(); with no parameters.
D-003	SpaceDemo.java, Line 46	for (int i = 0; i <= samples.length; i++) - Loop condition causes an ArrayIndexOutOfBoundsException on the final iteration.	Java Checklist: 4.2 (Bugs)	**Critical**	N/A	Change loop condition to i < samples.length.
D-004	SpaceDemo.java, Line 32	rocket.setPropulsion(ion); - This line will not compile because IonThrusterPropulsion does not implement the Propulsion interface.	Custom Checklist: 2.1 (Design)	**Critical**	N/A	Add implements Propulsion to IonThrusterPropulsion class.
D-005	Launchable.java, Line 3	Interface name is 1launchable. This is a Java syntax error; identifiers cannot start with a number.	Java Checklist: 1.1 (Syntax)	**Critical**	N/A	Rename interface to Launchable.
D-006	Maneuverable.java, Line 3	private execute = "maneuver"; - Interfaces cannot contain private fields. This is a Java syntax error.	Java Checklist: 1.1 (Syntax)	**Critical**	N/A	Remove the line.
D-007	ParachuteLanding.java, Line 15	return name+2=3; - The toString() method contains a syntax error (invalid assignment).	Java Checklist: 1.1 (Syntax)	**Critical**	N/A	Fix to return name;
High Severity Defects (Design & SOLID)						
D-008	Propulsion.java, Line 6	int getFuelLevel(); - The interface forces all implementing classes to have a getFuelLevel() method.	Checklist 1: 3.2 (ISP)	**High**	**ISP**	Remove getFuelLevel() from the Propulsion interface. It is an unrelated responsibility.
D-009	Landable.java, Line 3	The interface is empty. It provides no contract for the land() method.	Checklist 1: 3.1 (Design)	**High**	**ISP**	Add void land(); to the interface.
D-010	RelayCommunication.java, Line 8	transmitData(...) method has a different signature than the Communicable interface. It does not correctly implement the interface.	Custom Checklist: 2.1 (Design)	**High**	**LSP**	Change method signature to @Override public void transmitData(String payload).
D-011	Rocket.java, Line 23	setIonThruster(IonThrusterPropulsion ion) - The high-level Rocket class depends directly on the low-level concrete class IonThrusterPropulsion.	Checklist 1: 5.2 (DIP)	**High**	**DIP / OCP**	Remove this method. The SetPropulsion(Propulsion p) method should be used instead (after fixing its name).
D-012	SolidFuelPropulsion.java, Line 16	if (mode == "AUTO") - String comparison is performed using == instead of .equals().	Java Checklist: 4.4 (Bugs)	**High**	N/A	Change comparison to if ("AUTO".equals(mode)).
D-013	SpaceDemo.java, Line 31	if (ion.name == "IonThruster") - String comparison is performed using == instead of .equals().	Java Checklist: 4.4 (Bugs)	**High**	N/A	Change comparison to if ("IonThruster".equals(ion.name)).
D-014	Rocket.java, Line 51	if (propulsion == null) - A potential NullPointerException is checked, but execution is not stopped, causing a crash on the next line.	Java Checklist: 4.3 (Bugs)	**High**	N/A	if (propulsion == null) { throw new IllegalStateException("..."); }
Medium Severity Defects (Error Handling & SRP)						
D-015	SpaceDemo.java, Line 41	catch (Exception e) {} - An empty catch block is used, which silently swallows the FileInputStream error.	Checklist 3: 5.1 (Errors)	**Medium**	N/A	Log the exception (e.g., e.printStackTrace();) or handle it.
D-016	Rocket.java, Line 25	rocketCount++; - A setter method (setIonThruster) has a side effect of modifying a global static counter.	Checklist 1: 1.1 (SRP)	**Medium**	**SRP**	Remove this line. Counters should be managed by a factory or a separate class.
D-017	Rocket.java, Line 18	public static int rocketCount = 0; - Public, static, mutable field breaks encapsulation.	Checklist 2: 2.1 (Encap)	**Medium**	N/A	Make the field private or manage it via a factory.
D-018	SpaceDemo.java, Line 35	The else branch following if (magic > 0) is **dead code** because magic is a hardcoded constant (42).	Checklist 3: 7.1 (Unused)	**Medium**	N/A	Remove the unreachable else block.
Low Severity Defects (Naming & Style)						
D-019	Maneuverable.java, Line 2	Interface name maneuverable violates Java Class Naming Conventions (should be PascalCase).	Checklist 2: 1.1 (Naming)	**Low**	N/A	Rename to Maneuverable.
D-020	Rocket.java, Line 30	Method name SetPropulsion violates Java Method Naming Conventions (should be camelCase).	Checklist 2: 1.2 (Naming)	**Low**	N/A	Rename to setPropulsion.
D-021	SpaceDemo.java, Line 3	import java.util.*; - Use of wildcard imports is discouraged.	Checklist 2: 1.5 (Style)	**Low**	N/A	Use explicit imports (e.g., java.util.Scanner).
D-022	IonThrusterPropulsion.java, Line 6	public String name = "IonThruster"; - Public field breaks encapsulation.	Checklist 2: 2.1 (Encap)	**Low**	N/A	Make field private final and use toString() or a getter.

Given Checklist are filled are separate documents

4.2. Summary of Code Fixes

Following the inspection meeting, the team entered the **Rework** phase. All 22 defects identified in the Defect Log (Section 4.1) were assigned and resolved. The primary changes made to the Java source code are summarized below:

1. Resolved SOLID Principle Violations:

- **ISP (D-008):** The primary Interface Segregation Principle violation was fixed by removing the unrelated getFuelLevel() method from the Propulsion interface.
- **DIP / OCP (D-011):** The Rocket class was decoupled from concrete implementations by removing the setIonThruster() method, which violated the Dependency Inversion Principle. The class now relies only on the Propulsion interface, making it open for extension.
- **LSP (D-010):** The RelayCommunication class was refactored to correctly implement the Communicable interface by fixing its transmitData() method signature.

2. Fixed All Critical & High-Severity Defects:

- All compilation-blocking syntax errors were fixed (e.g., 1launchable, private execute = "maneuver", return name+2=3;).
- All critical runtime-crash bugs were eliminated, including the **divide-by-zero** error in Rocket.java (D-001) and the **ArrayIndexOutOfBoundsException** in SpaceDemo.java (D-003).
- All incorrect string comparisons using == were replaced with .equals() (D-012, D-013).
- The NullPointerException risk in Rocket.java's performManeuver() was fixed by adding a proper IllegalStateException check (D-014).

3. Enforced Interface Contracts:

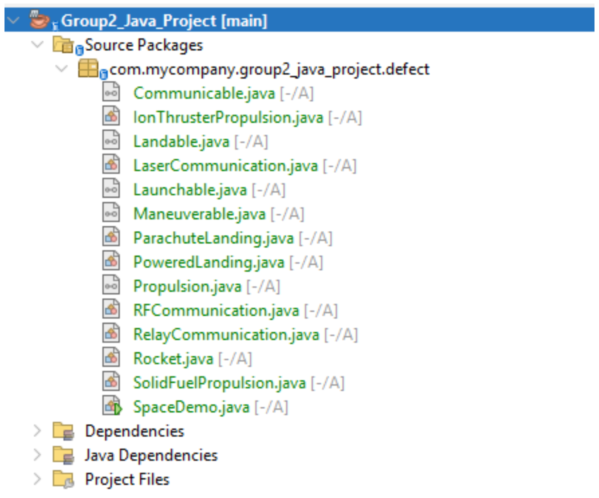
- The Landable interface contract was defined by adding the void land(); method (D-009).
- All implementing classes (ParachuteLanding, PoweredLanding) were updated to @Override this new, parameter-less land() method, resolving the critical compilation failure (D-002).
- IonThrusterPropulsion was modified to correctly implements Propulsion, allowing it to be used polymorphically (D-004).

4. General Code Quality & Best Practices:

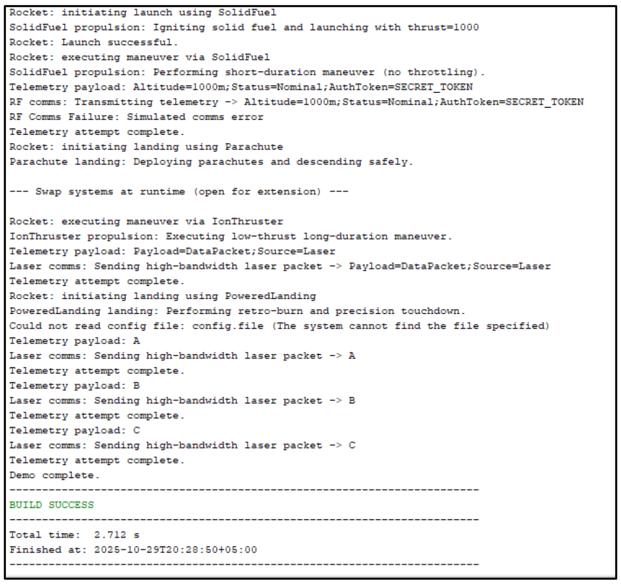
- All Java Naming Convention violations were fixed (e.g., SetPropulsion to setPropulsion, maneuverable to Maneuverable).
- Empty catch blocks were populated with e.printStackTrace() to ensure errors are logged (D-015).
- SRP violations, like the side effect in setIonThruster(), were removed (D-016).

The entire codebase is now in a stable, executable state that adheres to our fixed UML design.

5. Snapshot of Workable Project



```
Group2_Java_Project [main]
  Source Packages
    com.mycompany.group2_java_project.defect
      Communicable.java [-/A]
      IonThrusterPropulsion.java [-/A]
      Landable.java [-/A]
      LaserCommunication.java [-/A]
      Launchable.java [-/A]
      Maneuverable.java [-/A]
      ParachuteLanding.java [-/A]
      PoweredLanding.java [-/A]
      Propulsion.java [-/A]
      RFCommunication.java [-/A]
      RelayCommunication.java [-/A]
      Rocket.java [-/A]
      SolidFuelPropulsion.java [-/A]
      SpaceDemo.java [-/A]
  Dependencies
  Java Dependencies
  Project Files
```



```
Rocket: initiating launch using SolidFuel
SolidFuel propulsion: Igniting solid fuel and launching with thrust=1000
Rocket: Launch successful.
Rocket: executing maneuver via SolidFuel
SolidFuel propulsion: Performing short-duration maneuver (no throttling).
Telemetry payload: Altitude=1000m;Status=Nominal;AuthToken=SECRET_TOKEN
RF comms: Transmitting telemetry -> Altitude=1000m;Status=Nominal;AuthToken=SECRET_TOKEN
RF Comms Failure: Simulated comms error
Telemetry attempt complete.
Rocket: initiating landing using Parachute
Parachute landing: Deploying parachutes and descending safely.

--- Swap systems at runtime (open for extension) ---

Rocket: executing maneuver via IonThruster
IonThruster propulsion: Executing low-thrust long-duration maneuver.
Telemetry payload: Payload=DataPacket;Source=Laser
Laser comms: Sending high-bandwidth laser packet -> Payload=DataPacket;Source=Laser
Telemetry attempt complete.
Rocket: initiating landing using PoweredLanding
PoweredLanding landing: Performing retro-burn and precision touchdown.
Could not read config file: config.file (The system cannot find the file specified)
Telemetry payload: A
Laser comms: Sending high-bandwidth laser packet -> A
Telemetry attempt complete.
Telemetry payload: B
Laser comms: Sending high-bandwidth laser packet -> B
Telemetry attempt complete.
Telemetry payload: C
Laser comms: Sending high-bandwidth laser packet -> C
Telemetry attempt complete.
Demo complete.
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.712 s
Finished at: 2025-10-29T20:28:50+05:00
------------------------------------------------------------------------
```

6. Conclusion

This software inspection project provided a comprehensive, practical experience in the principles of Software Quality Engineering. By following a formal inspection process—from planning and preparation to inspection, rework, and follow-up—our team successfully identified and remediated 22 distinct defects from the provided codebase.

The inspection confirmed a direct correlation between poor design architecture and high defect density. The initial UML diagram and code were heavily flawed with violations of ***SOLID principles****, particularly the Interface Segregation Principle (ISP) and the Dependency Inversion Principle (DIP). These design-level issues were the root cause of many critical bugs, including compilation failures, runtime crashes (like the Divide by Zero error), and ArrayIndexOutOfBoundsExceptions.*

Through the Rework phase, we refactored the entire system to align with our new, SOLID-compliant UML design. This involved:

- *Breaking the "fat interface" into smaller, role-specific interfaces.*
- *Decoupling the high-level Rocket class from low-level concrete implementations.*
- *Fixing all code-level bugs and enforcing type safety.*

Ultimately, this project demonstrated that software quality is not merely about fixing bugs as they appear. It is about building a robust, maintainable, and extensible system from the ground up. The inspection process proved to be an invaluable tool for identifying these deep-rooted design flaws, and the SOLID principles provided the essential "map" to fix them correctly.